Task 2: Person Detection using RGB-D Sensor and Thermo Camera

M. Pecka, V. Kubelka, K. Zimmermann and M. Reinstein

March 27, 2014

1 Task 2 assignment

Doc. Ing. F. Zlo, CSc. wants to reduce the number of students by means of Robotron [1]. However, the publicly available OpenCV face detector he has been using so far produces too many false positive detections (e.g. printed faces on posters). Given the calibration from the previous task, you can augment bare RGB images with temperature and depth information. Avoid being exterminated by Doc. Ing. Zlo CSc., prove yourself useful and find a way to use the temperature and depth information to improve the detector so that it detects only faces of *real persons*. An example of how your algorithm should work is given in Figure 1. Please note that Doc. Ing. Zlo, who might appear in provided data, is not a *real person*.

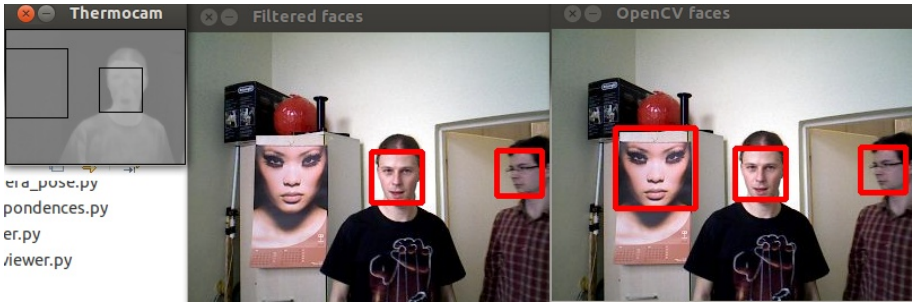

Figure 1: An example of detected faces filtering using the thermo camera data. Notice the face outside the thermo camera field of view – the thermo camera filter would have to leave it untouched even if it weren't a real person (you can't base your decision on missing data!).

1.1 Motivation

Except the use-case of Doc. Ing. Zlo, a precise detector of real persons can be further utilized in several ways. It may also serve as a part of a victim detection

system, which is essential for almost all Search&Rescue robots. Or it may be a part of a security video surveillance system. Last, but not least, it may be useful for interactive robots to detect people around them and start a conversation, offer help or some other task. Connected with (known) face recognition, the possibilities are even greater.

1.2 Provided datasets

Doc. Zlo provides you several BAG files with various "difficulty" levels, which means that some BAG files intentionally contain face-like objects to confuse the OpenCV detector. The topics recorded in the BAG files are the same as in Task 1. Along with these BAG files we provide you with the "ground truth" consisting of manually marked rectangles around faces (of real persons). The ground truth is saved in Matlab MAT files and has the same name as the corresponding BAG file. Each row of the MAT file contains a record of the type described by Table 1.

Timestamp.secs	Timestamp.nsecs	x_1	y_1	x_2	y_2

Table 1: The format of a row of the provided ground truth. First two columns together form the timestamp of the corresponding picture, and the x_1, y_1 values are coordintes of a face's top left corner, whereas x_2, y_2 represent the bottom right corner. For one timestamp, there can be more than one record (more faces in one picture) and there doesn't have to be any record for a given timestamp (no faces in the picture).

1.3 OpenCV detector construction

The OpenCV face detector is a sliding window detector based on Haar-like features, see [4] for details. Sliding window means that decisions about face presence/absence is made independently for each possible rectangular window in the image[1]. The decision is made by a classifier function $f(\mathbf{x})$ based on features $\mathbf{x}$ computed from the window content. The classifier solves binary decision task, with labels $\mathbf{y} \in \{\mathtt{F}, \mathtt{B}\}$ (i.e. Face and Background).

1.4 Evaluation of detector quality

In many applications, it is best to assign label `F` if the posterior probability $p(\mathtt{F}|\mathbf{x})$ of having face in the evaluated window (with features $\mathbf{x}$) is bigger than probability $p(\mathtt{B}|\mathbf{x})$ of having background in this window. The problem is that the face posterior probability strictly depends[2] on the target application. For

[1] Of course, if you slide a detection window over the image with the step equal to one pixel for all possible scales, there are too many windows to be evaluated, therefore *reasonable* step size must be chosen.

[2] via prior probability p(F)

example, the face probability is significantly smaller in a forest than in a pub. And even if the environment is the same, sometimes we want to detect all faces, and sometimes we want to avoid detecting false positives. To tackle this trade-off efficiently, we often follow the Bayes decision theory and replace the original rule

$$p(\mathtt{F}|\mathbf{x}) > p(\mathtt{B}|\mathbf{x}) \quad \Rightarrow \quad \mathtt{F} \tag{1}$$

with the more complicated one

$$\frac{p(\mathbf{x}|\mathtt{F})p(\mathtt{F})}{p(\mathbf{x})} > \frac{p(\mathbf{x}|\mathtt{B})p(\mathtt{B})}{p(\mathbf{x})} \quad \Rightarrow \quad \mathtt{F}. \tag{2}$$

Such statement is further simplified to

$$\frac{p(\mathbf{x}|\mathtt{F})}{p(\mathbf{x}|\mathtt{B})} > \frac{p(\mathtt{B})}{p(\mathtt{F})} = \theta \quad \Rightarrow \quad \mathtt{F}, \tag{3}$$

where the left fraction corresponds to the *likelihood ratio* and the right fraction is *prior probability ratio*, which is independent on features $\mathbf{x}$ computed in the evaluated window. We denote the prior ratio θ.

The classifier is function $f(\mathbf{x}) \approx \frac{p(\mathbf{x}|\mathtt{F})}{p(\mathbf{x}|\mathtt{B})}$ of features $\mathbf{x}$ trained to provide approximation of the likelihood ratio. The values of $f(\mathbf{x})$ are called the classifier *response* to the features $\mathbf{x}$. Having the classifier at hand, we set the prior ratio θ according to the requirements of our target application. The higher is θ, the smaller is the number of detected true faces (called *true positive rate* `TP`) but also the smaller is the number of false detections (called *false positive rate* `FP`). A graph depicting the relation between $\mathtt{TP}(\theta)$ and $\mathtt{FP}(\theta)$ as a function of θ is called *ROC curve*, see Figure 2. ROC curve determines the quality of the classifier – for example an ideal classifier would have some threshold θ for which we would have $\mathtt{TP}(\theta) = 1$ and $\mathtt{FP}(\theta) = 0$, i.e. detect 100% faces and make no mistake. However, it is not the case in real world problems and we have to choose from the trade-off between `TP` and `FP`.

1.5 OpenCV detector parameters

Unlike other face detectors, the OpenCV face detector doesn't provide a simple way to specify θ. However, to draw the ROC curve for the detector, you need a parameter you can change to observe the changes in `TP` and `FP`. As an alternative, OpenCV provides two parameters to its method `detectMultiScale` that influence the number of detections.

The first is called `scaleFactor`. Its meaning is the following. The detector starts finding faces with a large window (of size `maxSize`). When it is finished with searching using this large window, it applies the `scaleFactor` to the window size to get the new window size. The search stops when the window size is `minSize`.

The second parameter is `minNeighbors` and it denotes the minimum number of overlapping rectangles to mark a rectangle as a face. This is based on the

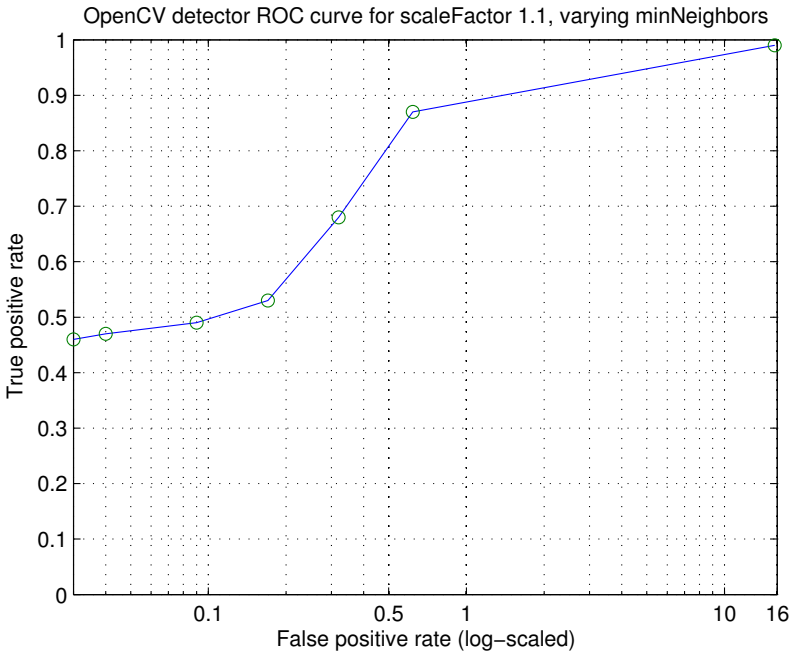

Figure 2: An example ROC curve. This one was evaluated on different data than you are provided with, so its shape may differ from the one you get.

fact that every face can be detected with several windows of consecutive window sizes. So if we have e.g. a detection only for a single window size, we can assume the detection is a false alarm. On the other hand, if `scaleFactor` is not high (e.g. near 1.1), having about 3–5 detections of different sizes (but with similar centers) increases the probability that this cluster of rectangles represents a face. After filtering out the false detections (with cluster sizes smaller than `minNeighbors`) the detector selects the most probable faces from all the remaining clusters.

Putting these two together, you can control the number of faces the detector returns. Although it is not equivalent to setting θ, you can play with these two parameters to obtain a ROC curve for the OpenCV detector.

2 The task to be solved

The steps mentioned below should guide you through the completion of Task 2.

1. Put the template code to a working state

   - Download the code template from CourseWare [3] and also download the datasets belonging to this task. Extract the datasets e.g. to

`task2/data`.

- Fill in the places in the code marked with the `# IMPLEMENT` comment to get the code working. You'll need to write your own subscribers to the relevant topics and you will also need to create a publisher for the results. When creating the publisher, decide on the type of the published message to be used, and if needed, create a new type of ROS message (do not forget to import it in the Python code and to `rosmake` the whole project).
- When you have the base of the code working, you should be able to call `rosrun task2 face_detector.py` and while playing a dataset BAG file, the program should display images from the BAG file with the detected faces drawn as red rectangles.
- If you inspect the code thoroughly, you will discover some threading-related code. We provided it to you to simplify and safeguard the multi-image synchronization task.

2. Implement the filtering

   - Implement the face filtering. It is performed in method `filter_face`, which takes one face and returns `True` if the face belongs to a real person.
   - There are several methods to filter the detected faces based on the sensor data. You are required to come up with (at least) two methods that take into account the temperature and the depth information. You are only required to filter the face detections provided by the OpenCV face detector.
   - By no means are you required to develop a face detection algorithm yourself. In fact, you're not allowed to use any other face detection algorithms than the OpenCV algorithm provided in the code template.
   - It is probable that your code will no longer be able to process data in the 3 Hz frequency as the data come. You can do three things to cope with this situation. You can set the `queue_size` parameter when creating the subscribers, which means the delayed messages for that subscriber are buffered. Or you can utilize the `rate` parameter of `rosbag play`, which slows down the BAG file playback. Last, you can use school computers which may be faster than your computer (the lab is usually not full whole day – if you come at any time, it is highly probable you can ask the teacher in the room, sit to a computer and silently work on Task 2).

3. Publish the filtered faces

   - As the result of this task we expect you to publish messages to a ROS topic containing the detected and filtered faces of real people in the scene.

- It is left up to you to decide what type of ROS message to use for that (maybe you would need to create your own) and what should be the name of the topic on which you publish your results.
- Do not forget to document these decisions in the report.

4. Play with the parameters and draw ROC curves

   - It is highly recommended to draw the ROC curves to see the influence of your filter, otherwise you can easily do changes which look reasonable but harm the overall detection rate.
   - You are required to draw the ROC curve into the report for (i) the provided OpenCV face detector with parameters used in your implementation, and (ii) your improved detector (using your filtering; choose a single parameter of your filters to be the changing variable).
   - The highest achieved `TP` for `FP` $= 0.05$ will be rewarded by a *bottled beer signed by all IRO teachers and personal congratulation of doc. Ing. Zlo, CSc. in front of the whole classroom.*

   Note, that all these ROC curves are a compulsory part of your report. Both `TP` and `FP` are relative with respect to *all* ground truth faces in all BAG files – for example: if you have 10 incorrect detections and 50 correct detections out of 100 ground truth faces, then `TP` $= 0.5$ and `FP` $= 0.1$). This also means `TP` can not have values greater than 1 (the best we can do is detect all faces), but `FP` can surely go beyond 1 (if the detector shows too much false faces). A correctly detected face is a bounding box B covering ground truth bounding box B_0 by more than 50%, i.e.

$$\frac{B \cap B_0}{B \cup B_0} \geq 0.5 \tag{4}$$

5. Write the report

   - Except the requirements on the report given at the CourseWare page, add some more information about your approach.
   - Describe the process how you decided for the specific ROS message type you chose for the result messages.
   - Describe how you evaluate the quality of your filters and try to give an estimate on how you think it is good on an absolute scale.
   - Give some thoughts about how could your algorithm be improved – be it e.g. by larger training dataset or incorporating data from more sensors.
   - Provide a set of images on which your algorithm works well, and find some images on which the algorithm doesn't behave well. Try to explain why it happens.

- Take the best detector settings you have found and express the probability that your detector says "not a real person" while there actually is a real person. The mathematical background for this part was presented on the first IRO lecture [2]. Discuss the relation of this value to the ROC curves in your report.

6. Submit your work

   - Your work has to be uploaded to the Upload system. The report is submitted to task `2_report`. All your codes should be zipped and submitted to task `2_detect`. Do not include any binary files in the submitted package (BAG files, MAT files...).

References

[1] Bugemos. Robotron. http://bugemos.com/komiksy/Student/2-10.gif.

[2] Michal Reintein. Iro lecture 1 slides. https://cw.felk.cvut.cz/wiki/_media/courses/a3m33iro/01_iro_probability.pdf.

[3] IRO Team. Task 2 cw page. https://cw.felk.cvut.cz/wiki/courses/a3m33iro/tasks/task2.

[4] P. Viola and M. Jones. Robust real-time face detection. In *ICCV*, volume 2, pages 747–757, 2001.